Graph Algorithms

Minimum Spanning Trees

Michael Lampis

November 13, 2025

The story so far

- (Short) Reachability problems:
  - Is there a path from s to t? (Strongly) connected components? ...
  - What is the shortest path from s to t? From everyone to everyone?

The story so far

- (Short) Reachability problems:
  - Is there a path from s to t? (Strongly) connected components? ...
  - What is the shortest path from s to t? From everyone to everyone?
- Algorithms:
  - Unweighted Reachability: BFS/DFS $O(m+n)$ time
  - Single-Source Shortest Paths:
    - Unweighted: BFS $O(m+n)$ time
    - Positive weights: Dijkstra $O((m+n)\log n)$ time (with min-heaps)
    - General weights: Bellman-Ford $O(mn)$ time

The story so far

- (Short) Reachability problems:
  - Is there a path from s to t? (Strongly) connected components? ...
  - What is the shortest path from s to t? From everyone to everyone?
- Algorithms:
  - Unweighted Reachability: BFS/DFS $O(m+n)$ time
  - Single-Source Shortest Paths:
    - Unweighted: BFS $O(m+n)$ time
    - Positive weights: Dijkstra $O((m+n)\log n)$ time (with min-heaps)
    - General weights: Bellman-Ford $O(mn)$ time
- New problem: Minimum Weight Spanning Tree
  - Find minimum weight set of edges that maintains connectedness.

Problem Definition

Definition

Definition

For a connected, edge-weighted graph $G = (V, E)$, a **Minimum Weight Spanning Tree** is a set of edges $E' \subseteq E$ such that

1. $G = (V, E')$ is connected.
2. The weight of E' is minimum among sets satisfying (1).

Definition

Definition

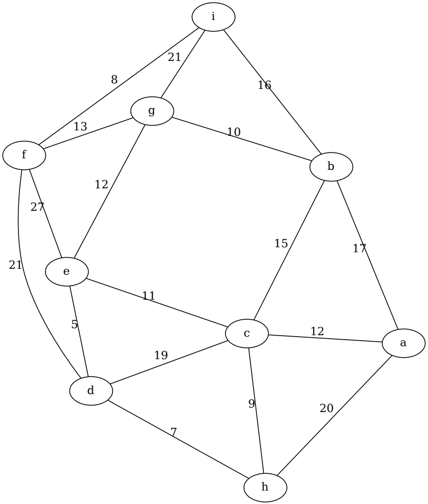
For a connected, edge-weighted graph $G = (V, E)$, a **Minimum Weight Spanning Tree** is a set of edges $E' \subseteq E$ such that

1. $G = (V, E')$ is connected.
2. The weight of E' is minimum among sets satisfying (1).

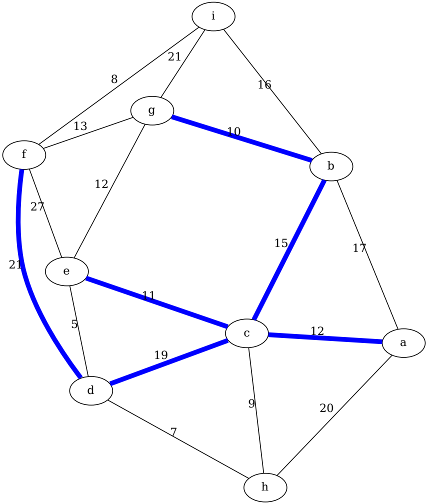
Two keywords:

- Tree: a connected graph without cycles.
- Spanning: a subgraph touching all vertices.

Example

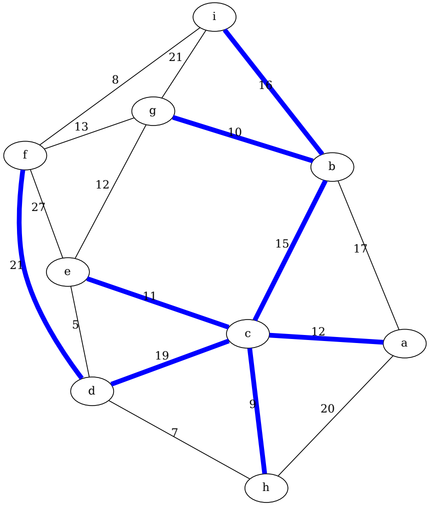


Example



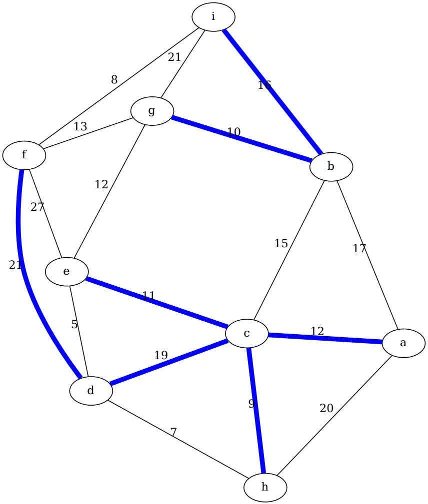
This is a tree, but it is **not** spanning.

Example



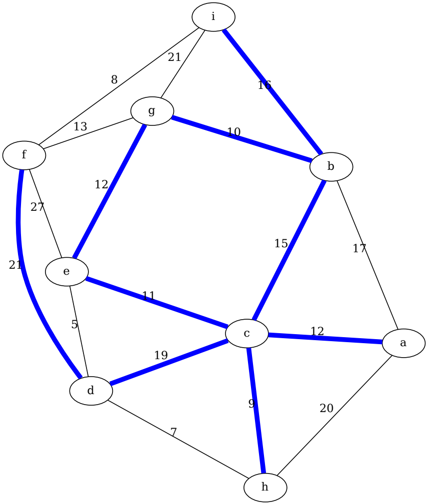
This is a tree, and it is spanning.

Example



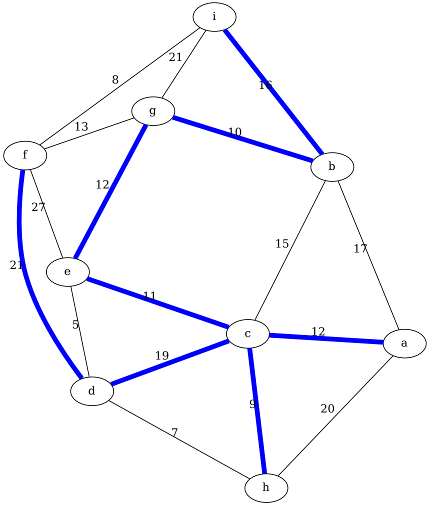
This is **not** a tree (not connected), it is spanning.

Example



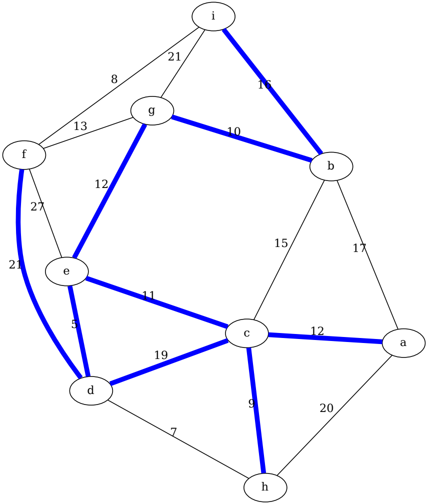
This is **not** a tree (contains cycle), it is spanning.

Example



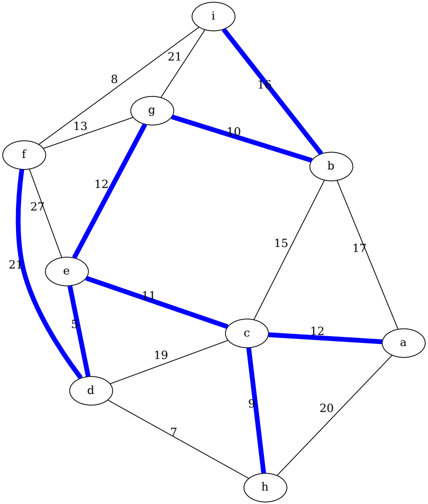
This is better than the previous spanning tree (exchanged weight 15 with weight 12). Can we do better?

Example



Take a cheap edge, add it to the solution (now we don't have a tree any more).

Example



Remove something more expensive from the resulting cycle.

Trees

Definition

A **tree** is a connected acyclic undirected graph.

Trees

Definition

A **tree** is a connected acyclic undirected graph.

- The above is the standard definition in graph theory.
- In computer science, we often deal with **rooted** trees
  - One special vertex is called the root.
  - All vertices have exactly one parent, except the root which has none.
- The two definitions are equivalent, if we don't care about a specific root (which we don't in the MST context).

Trees

Definition

A **tree** is a connected acyclic undirected graph.

- The above is the standard definition in graph theory.
- In computer science, we often deal with **rooted** trees
  - One special vertex is called the root.
  - All vertices have exactly one parent, except the root which has none.
- The two definitions are equivalent, if we don't care about a specific root (which we don't in the MST context).
- A disconnected union of trees is called a **forest**.

Basic properties

Lemma

If C is a cycle of $G = (V, E)$ and $e \in C$ an edge of C, then $G - e$ has the same connected components as G.

Lemma

In a tree, there is a unique path between any two vertices.

Lemma

A tree on n vertices has exactly $n - 1$ edges.

Lemma

A tree on n vertices has exactly $n-1$ edges.

Lemma

A tree on n vertices has exactly $n-1$ edges.

Proof.

More strongly: (i) A forest on n vertices with $c \leq n-1$ edges has exactly $n-c$ components (ii) A tree cannot have $\geq n$ edges.

- (i) For $c = 0$ true: all vertices are in distinct components.
- (i) Inductive step: removing an edge increases the number of components by 1.
- (ii) Take a minimum counter-example graph G, take an edge e
  - If $e = xy$ is not a bridge: there is another $x \to y$ path, we have a cycle, contradiction.
  - If e is a bridge, $G - e$ has two components C_1, C_2, and $\geq |C_1| + |C_2| - 1$ edges, so one component is a smaller forest with too many edges, contradiction.

□

Tree characterizations

Theorem

Consider the following three properties of an undirected graph G:

1. *G is acyclic*
2. *G is connected*
3. *G has $n - 1$ edges*

Any graph that satisfies two of these properties, must also satisfy the third.

Tree characterizations

Theorem

Consider the following three properties of an undirected graph G:

1. *G is acyclic*
2. *G is connected*
3. *G has $n-1$ edges*

Any graph that satisfies two of these properties, must also satisfy the third.

Proof.

See you next year for a full proof of this and other characterizations of the class of trees! □

Prim's Algorithm

Outline

- Input: An undirected, connected, edge-weighted graph G (could have negative weights).
- Output: A spanning tree of G of minimum weight.

Outline

- Input: An undirected, connected, edge-weighted graph G (could have negative weights).
- Output: A spanning tree of G of minimum weight.

- Strategy: greedy algorithm
- Start from an arbitrary vertex s, maintain a tree T containing s
- At every step add to the tree the vertex that is **closest** to T

Outline

- Input: An undirected, connected, edge-weighted graph G (could have negative weights).
- Output: A spanning tree of G of minimum weight.

- Strategy: greedy algorithm
- Start from an arbitrary vertex s, maintain a tree T containing s
- At every step add to the tree the vertex that is **closest** to T

Compare to Dijkstra's algorithm

- At every step add to the tree the vertex that is **closest** to s

Prim's algorithm

```
Initialize (dist[1...n] = ∞, parent[1...n] ← NULL, etc. )
Initialize Q← ∅
Q.Insert(s,0)
while Q not empty do
    u ←Q.Extract()
    for v ∈ N⁺(u) do
        if dist[v]>dist[u]+w(u,v) then                ▷ Shortcut found
            Parent[v] ← u
            dist[v] ← dist[u]+w(u,v)
            Q.Decrease(v,dist[v])
        end if
    end for
end while
```

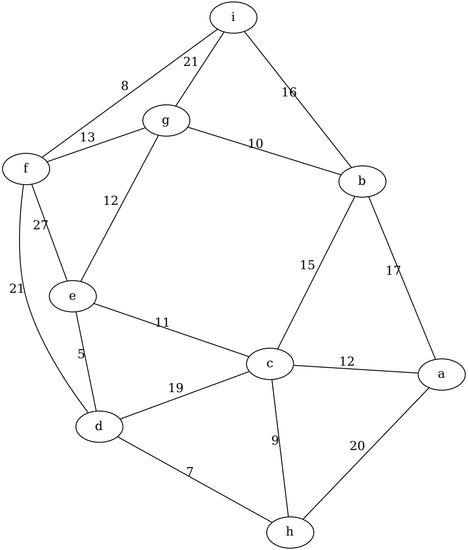
Prim's algorithm

```
Initialize (dist[1...n] = ∞, parent[1...n] ← NULL, etc. )
Initialize Q← ∅ ,T ← {s}
Q.Insert(s,0)
while Q not empty do
    u ←Q.Extract() ,T ← T ∪ {u}
    for v ∈ N⁺(u) \ T do
        if dist[v]> w(u, v)    then                    ▷ Shortcut found
            Parent[v] ← u
            dist[v] ←  w(u, v)
            Q.Decrease(v,dist[v])
        end if
    end for
end while
```

Dijkstra to Prim

- General strategy:
  - For Dijkstra: keep track of distances from s
  - For Prim: keep track of distances from T, the tree that contains s
- Output:
  - For Dijkstra: shortest-path tree from s
  - For Prim: Minimum-weight Spanning Tree (MST) (independently of choice of s)
- Complexity:
  - For both: $O(m+n)$ Priority Queue Operations
  - Priority Queue implemented via Min-Heap $\Rightarrow O(\log n)$ per operation
  - Time complexity: $O((m+n)\log n)$.
  - (As usual, assuming arithmetic operations are $O(1)$ time...)

Example



Distances:

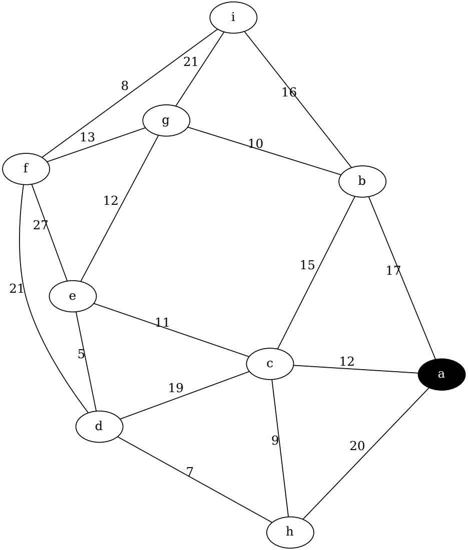
a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(0, a)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

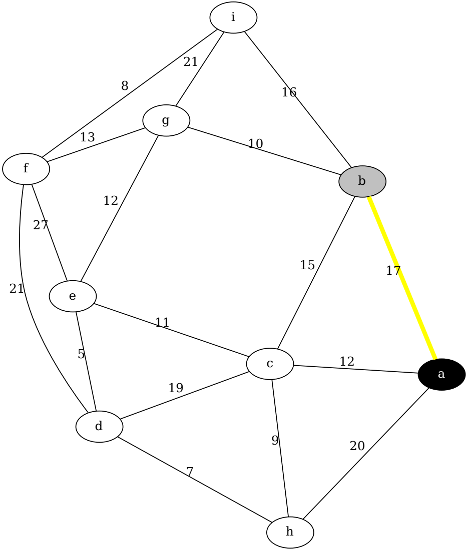
a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(0, a)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

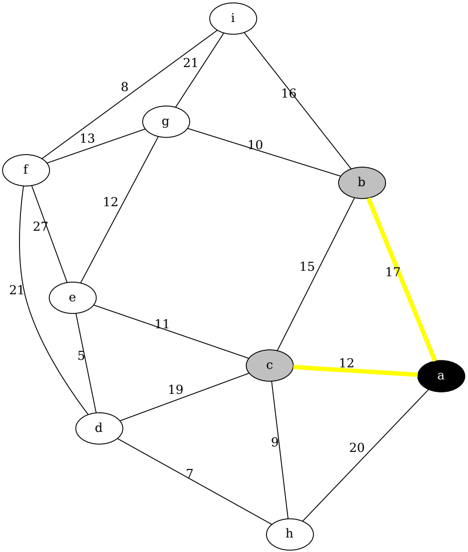
a	0
b	17
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(17, b)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

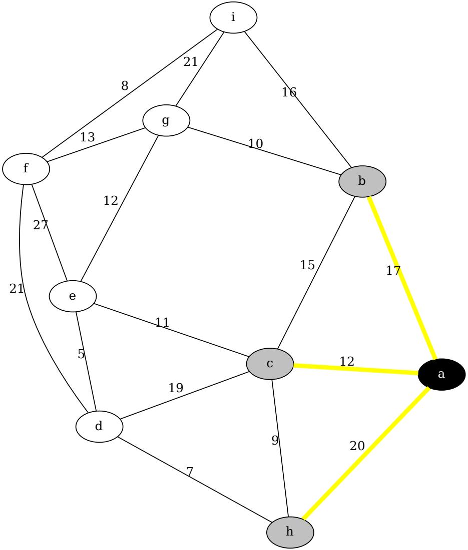
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	∞
i	∞

Priority Queue:
(12, c), (17, b)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

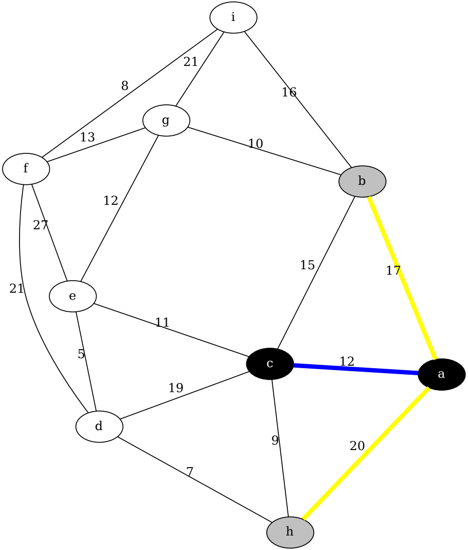
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
$(12, c), (17, b), (20, h)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

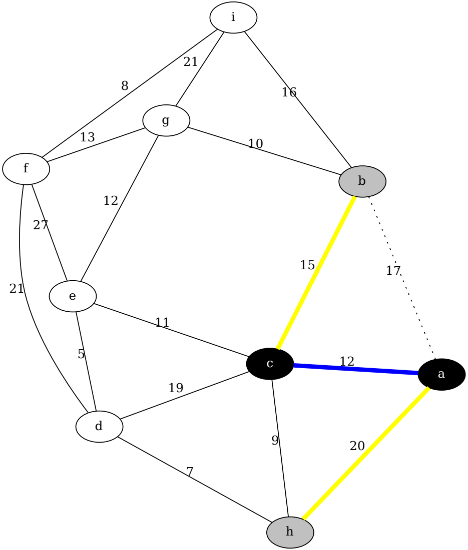
a	0
b	17
c	12
d	∞
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
(17, b), (20, h)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

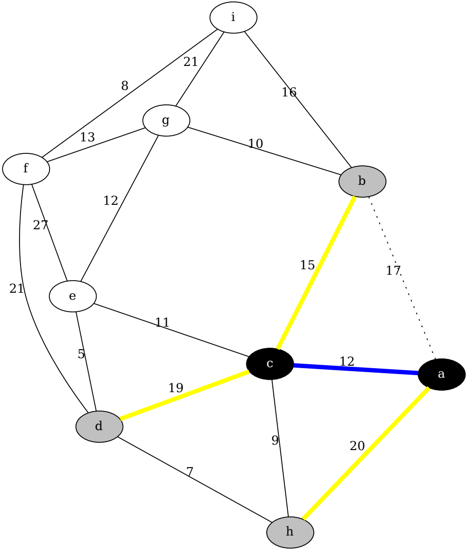
a	0
b	15
c	12
d	∞
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
(15, b), (20, h)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

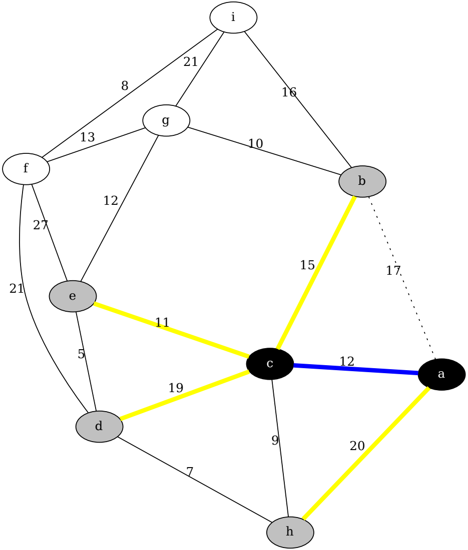
a	0
b	15
c	12
d	19
e	∞
f	∞
g	∞
h	20
i	∞

Priority Queue:
(15, b), (19, d), (20, h)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

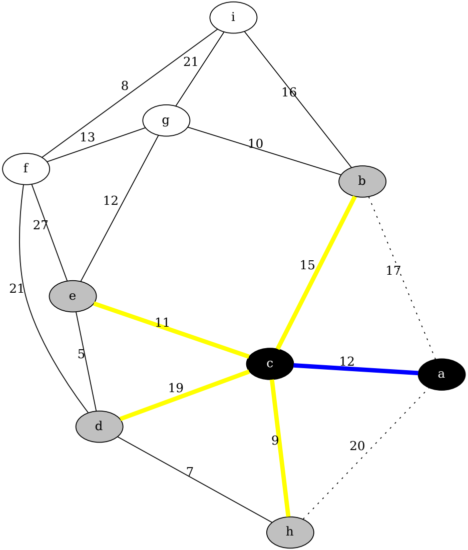
a	0
b	15
c	12
d	19
e	11
f	∞
g	∞
h	20
i	∞

Priority Queue:
$(11, e)$, $(15, b)$, $(19, d)$, $(20, h)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

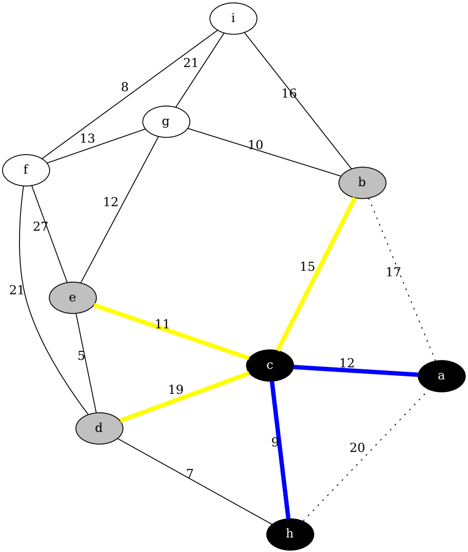
a	0
b	15
c	12
d	19
e	11
f	∞
g	∞
h	9
i	∞

Priority Queue:
(9, h), (11, e), (15, b), (19, d)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



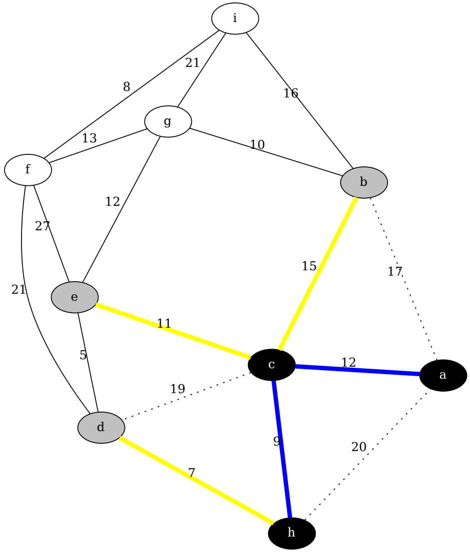
Distances:	
a	0
b	15
c	12
d	19
e	11
f	∞
g	∞
h	9
i	∞

Priority Queue:
$(11, e), (15, b), (19, d)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



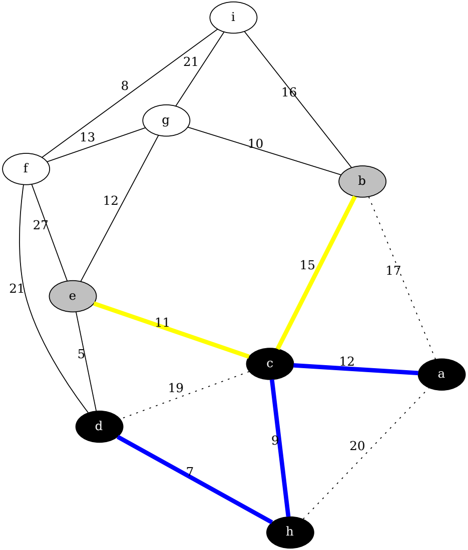
Distances:	
a	0
b	15
c	12
d	7
e	11
f	∞
g	∞
h	9
i	∞

Priority Queue:
$(7, d), (11, e), (15, b)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



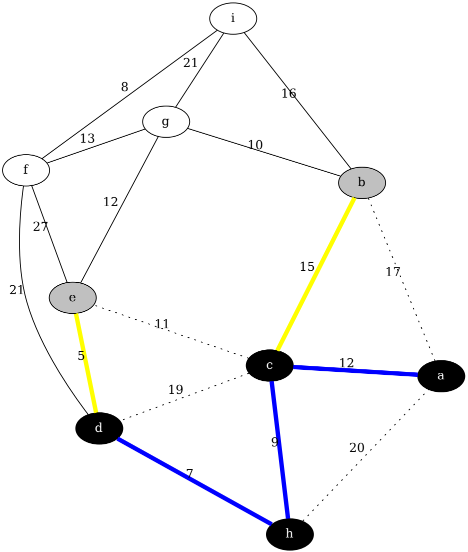
Distances:	
a	0
b	15
c	12
d	7
e	11
f	∞
g	∞
h	9
i	∞

Priority Queue:
(11, e), (15, b)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



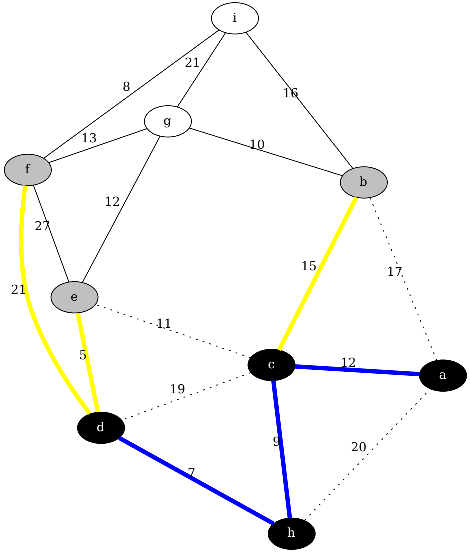
Distances:	
a	0
b	15
c	12
d	7
e	5
f	∞
g	∞
h	9
i	∞

Priority Queue:
(5, e), (15, b)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

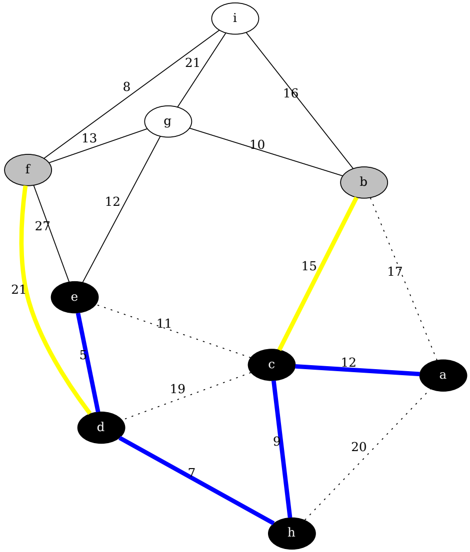
a	0
b	15
c	12
d	7
e	5
f	21
g	∞
h	9
i	∞

Priority Queue:
$(5, e)$, $(15, b)$, $(21, f)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



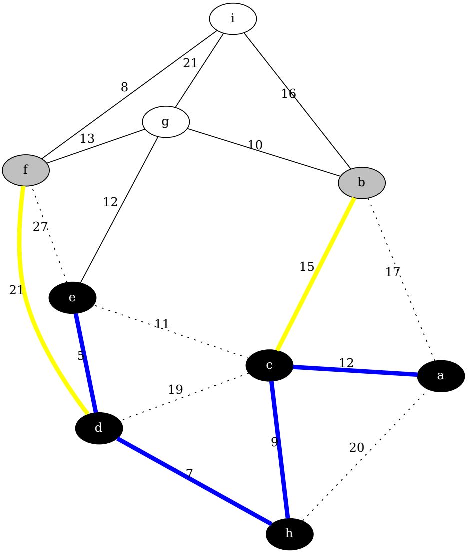
Distances:	
a	0
b	15
c	12
d	7
e	5
f	21
g	∞
h	9
i	∞

Priority Queue:
(15, b), (21, f)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

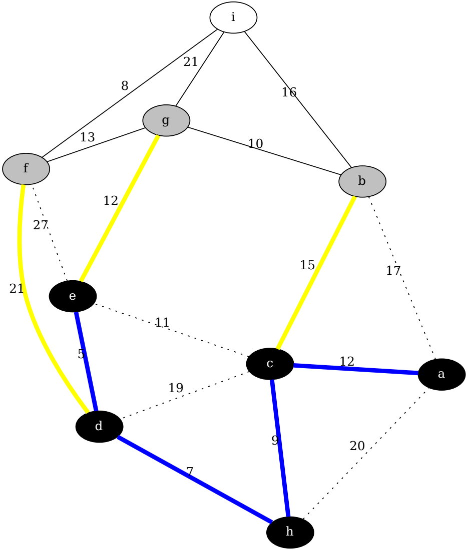
a	0
b	15
c	12
d	7
e	5
f	21
g	∞
h	9
i	∞

Priority Queue:
(15, b), (21, f)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



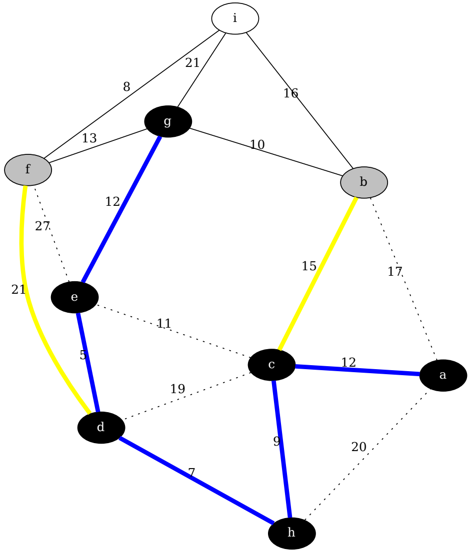
Distances:	
a	0
b	15
c	12
d	7
e	5
f	21
g	12
h	9
i	∞

Priority Queue:
$(12, g), (15, b), (21, f)$

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

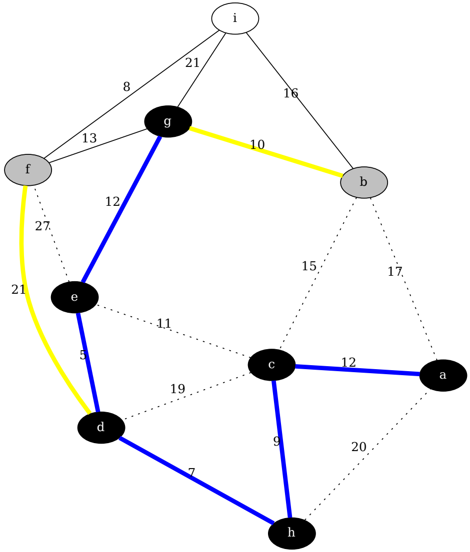
a	0
b	15
c	12
d	7
e	5
f	21
g	12
h	9
i	∞

Priority Queue:
$(15, b), (21, f)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

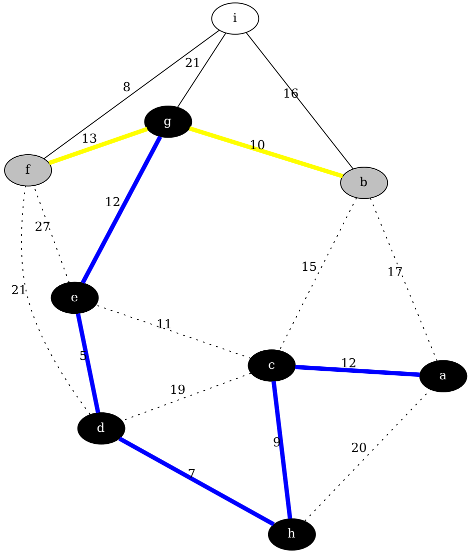
a	0
b	10
c	12
d	7
e	5
f	21
g	12
h	9
i	∞

Priority Queue:
$(10, b), (21, f)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

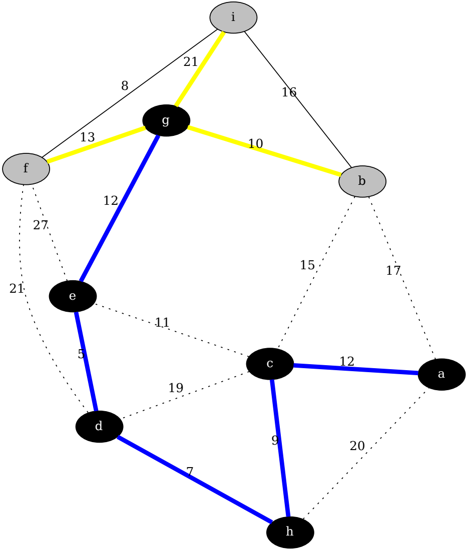
a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	∞

Priority Queue:
(10, b), (13, f)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

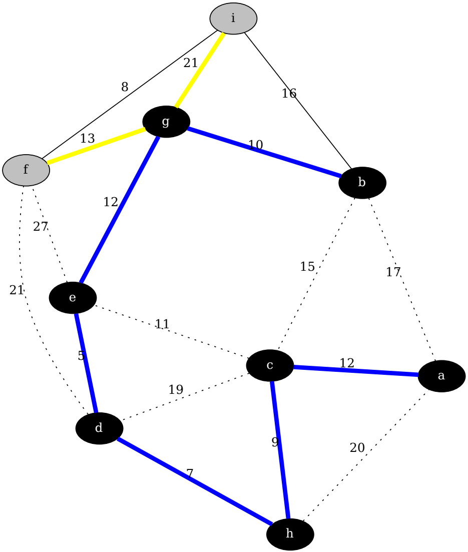
a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	21

Priority Queue:
(10, b), (13, f), (21, i)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

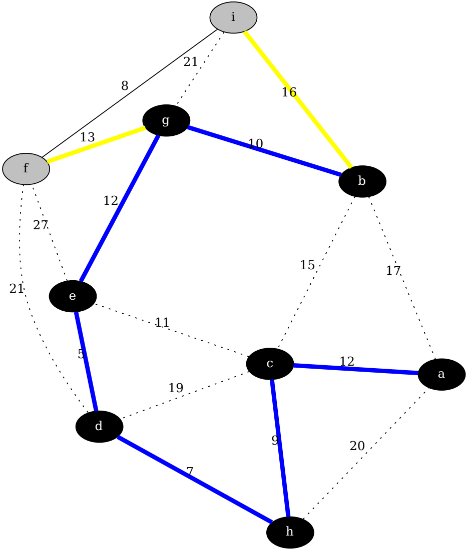
a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	21

Priority Queue:
$(13, f), (21, i)$

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

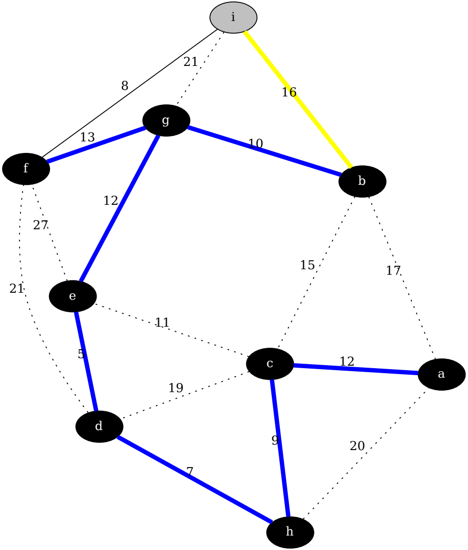
a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	16

Priority Queue:
(13, f), (16, i)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

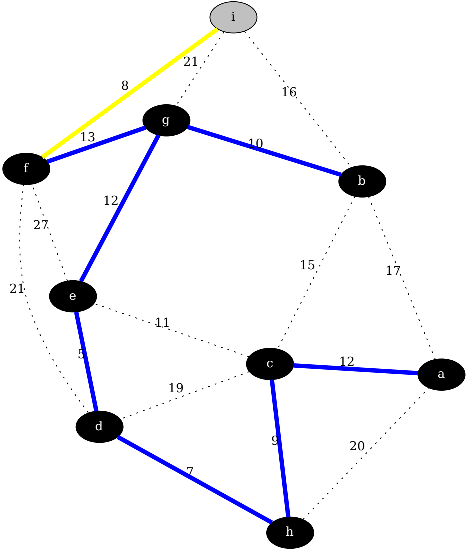
a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	16

Priority Queue:
(16, i)

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



Distances:

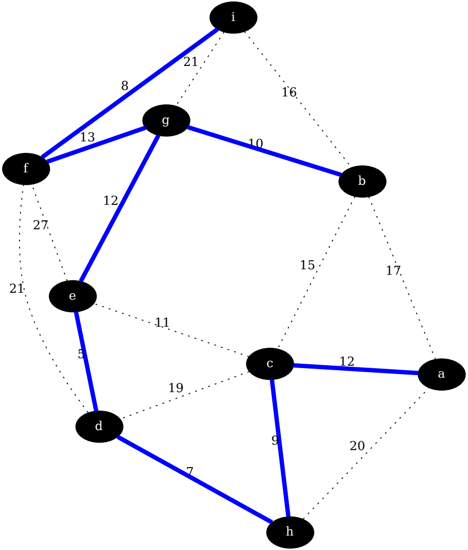
a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	8

Priority Queue:
(8, i)

Legend:

- White vertex $\rightarrow$ no path found.
- Gray vertex $\rightarrow$ some path found.
- Black vertex $\rightarrow$ vertex is in T.
- Black edge $\rightarrow$ not yet considered.
- Blue edge $\rightarrow$ tree edge.
- Yellow edge $\rightarrow$ considered edge.
- Dotted edge $\rightarrow$ non-optimal edge.

Example



Distances:

a	0
b	10
c	12
d	7
e	5
f	13
g	12
h	9
i	8

Priority Queue:
$\emptyset$

Legend:

- White vertex $\longrightarrow$ no path found.
- Gray vertex $\longrightarrow$ some path found.
- Black vertex $\longrightarrow$ vertex is in T.
- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Yellow edge $\longrightarrow$ considered edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Proof of correctness

High-level strategy:

- Prim's algorithm definitively adds an edge to the tree each time we extract a vertex from the queue.
- Easier to see: Prim produces a spanning tree, because we add $n-1$ edges, produce a tree rooted at the initial vertex.
- Induction: assuming there exists an optimal MST containing the first i edges added, there exists an optimal solution also containing the first $i+1$ edges added.
- Why is the new edge optimal?

Light edges

Definition

Let $G = (V, E)$ be an edge-weighted undirected graph, $S \subseteq V$, and $e \in E$ such that e has exactly one endpoint in S. Then, e is a **light** edge for set S if it has minimum weight among all edges with exactly one endpoint in S.

Light edges

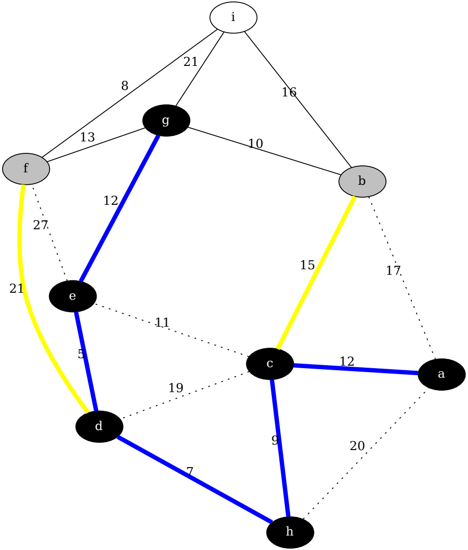
Definition

Let $G = (V, E)$ be an edge-weighted undirected graph, $S \subseteq V$, and $e \in E$ such that e has exactly one endpoint in S. Then, e is a **light** edge for set S if it has minimum weight among all edges with exactly one endpoint in S.

Lemma

Let G, S, e as above and e a light edge for S. Let A be a set of edges with 0 or 2 endpoints in S such that $A \subseteq T^*$, for some optimal MST T^*. Then, there exists an optimal MST that contains $A \cup \{e\}$.

Example



- $S = \{a, c, d, e, g, h\}$
- Blue edges are contained in some optimal solution
- Edge bg is **light** for S
- $\Rightarrow$ safe to add to current solution.

Lemma

Let G, S, e as previously and e a light edge for S. Let A be a set of edges with 0 or 2 endpoints in S such that $A \subseteq T^*$, for some optimal MST T^*. Then, there exists an optimal MST that contains $A \cup \{e\}$.

Lemma

Let G, S, e as previously and e a light edge for S. Let A be a set of edges with 0 or 2 endpoints in S such that $A \subseteq T^*$, for some optimal MST T^*. Then, there exists an optimal MST that contains $A \cup \{e\}$.

Proof.

- If T^* contains e, done!
- Otherwise, let $e = xy$ with $x \in S, y \notin S$. Add e to T^*.
- We have a cycle, which must use another edge e' with one endpoint in S.
- $\Rightarrow e' \notin A$
- $w(e') \geq w(e)$ as e is light.
- Remove e' to obtain a better (or equally good) spanning tree that contains $A \cup \{e\}$

$\square$

Theorem

Prim's algorithm is correct.

Theorem

Prim's algorithm is correct.

Proof.

Claim: every edge added is light for the set of vertices incident on previously selected vertices.

- Let T_i be the tree we have after i iterations.
- Suppose we now add edge xy, with x just extracted from queue, but edge $x'y'$ has $w(x'y') < w(xy)$, with $x, x' \in T_i$ and $y, y' \notin T_i$.
- We have extracted y' in some previous iterations, so the distance we have calculated for x' is $w(x'y') < w(xy)$, which is the distance we have for x. This contradicts the selection of x (we should have extracted x').

Theorem

Prim's algorithm is correct.

Proof.

Claim: every edge added is light for the set of vertices incident on previously selected vertices.

- Let T_i be the tree we have after i iterations.
- Suppose we now add edge xy, with x just extracted from queue, but edge $x'y'$ has $w(x'y') < w(xy)$, with $x, x' \in T_i$ and $y, y' \notin T_i$.
- We have extracted y' in some previous iterations, so the distance we have calculated for x' is $w(x'y') < w(xy)$, which is the distance we have for x. This contradicts the selection of x (we should have extracted x').

To complete the proof, apply the lemma $n - 1$ times. $\square$

Kruskal's Algorithm

Kruskal's Algorithm

- Solves the same problem as Prim's algorithm (MST)
- Also, greedy strategy.
- **Difference:** instead of growing a tree, we greedily grow a **forest**, at each step adding the least expensive edge.
- "Comparable" complexity to Prim (more on this later)

Kruskal's algorithm

```
Sort edges e_1, ..., e_m in increasing order of weight.
T ← ∅
for i = 1 to m do
    if T ∪ {e_i} has no cycle then
        T ← T ∪ {e_i}
    end if
end for
Output T
```

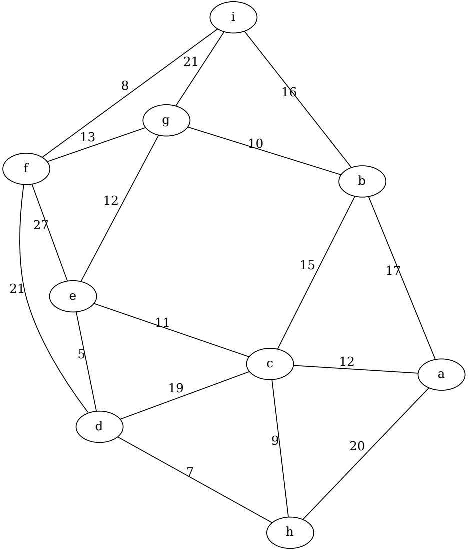
Kruskal's algorithm

```
Sort edges e_1, ..., e_m in increasing order of weight.
T ← ∅
for i = 1 to m do
    if T ∪ {e_i} has no cycle then        ▷ How to check this?
        T ← T ∪ {e_i}
    end if
end for
Output T
```

Kruskal's algorithm

Sort edges $e_1, \ldots, e_m$ in increasing order of weight.
$T \leftarrow \emptyset$
for $i = 1$ to m **do** $\triangleright$ Alternatively, end when T has $n-1$ edges
  if $T \cup \{e_i\}$ has no cycle **then** $\triangleright$ How to check this?
    $T \leftarrow T \cup \{e_i\}$
  end if
end for
Output T

Example



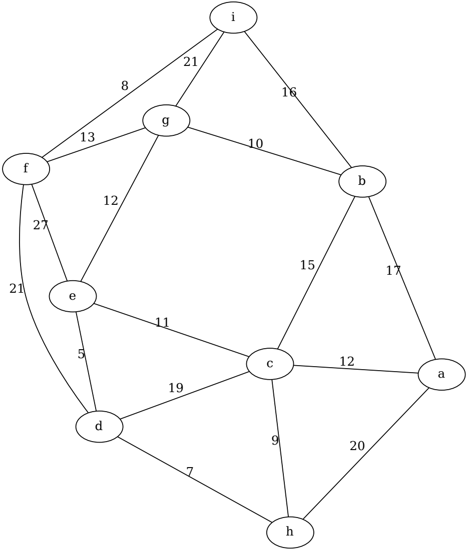
Sorted list of edges:

Edge	Weight
de	5
dh	7
fi	8
ch	9
bg	10
ce	11
ac	12
eg	12
fg	13
bc	15
bi	16
ab	17
cd	19
ah	20
df	21
gi	21
ef	27

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



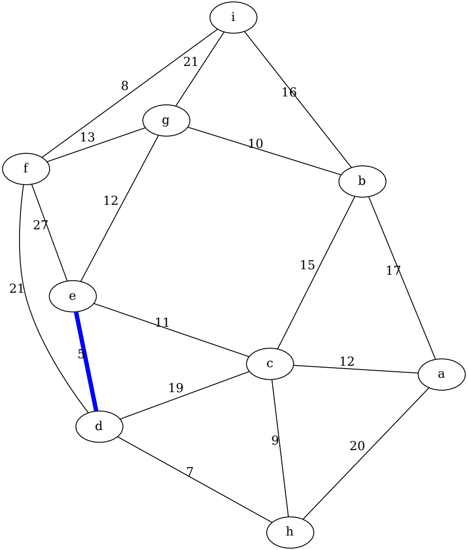
Sorted list of edges:

Edge	Weight	
de	5	←
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



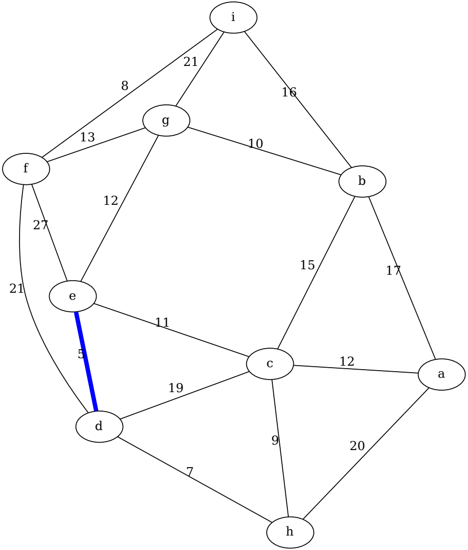
Sorted list of edges:

Edge	Weight	
de	5	←
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



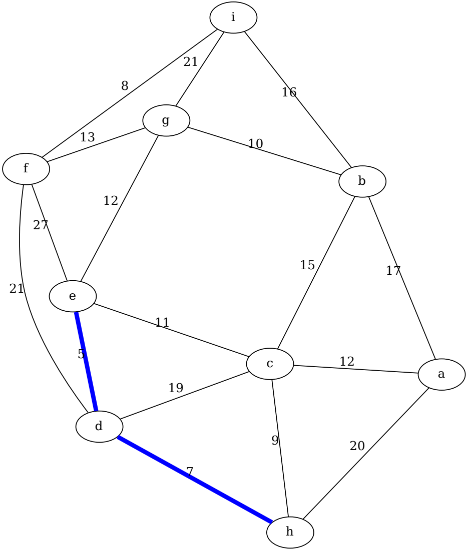
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	←
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



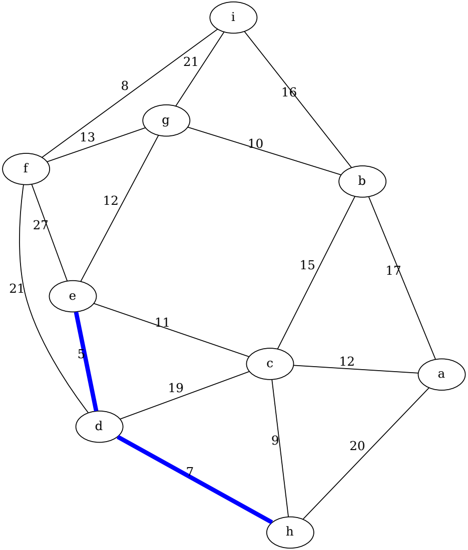
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	←
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



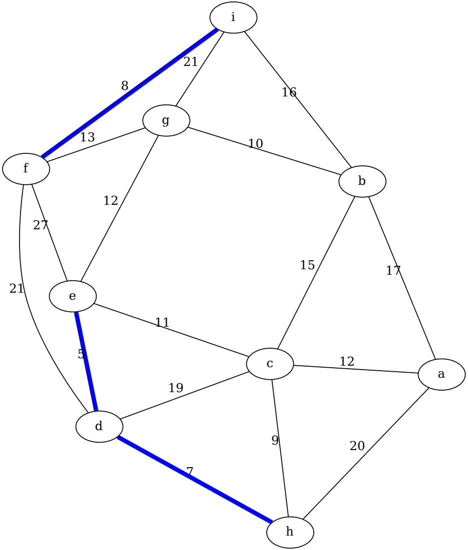
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	←
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



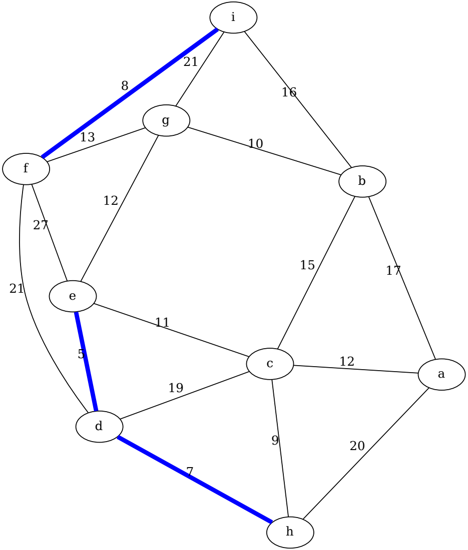
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	←
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



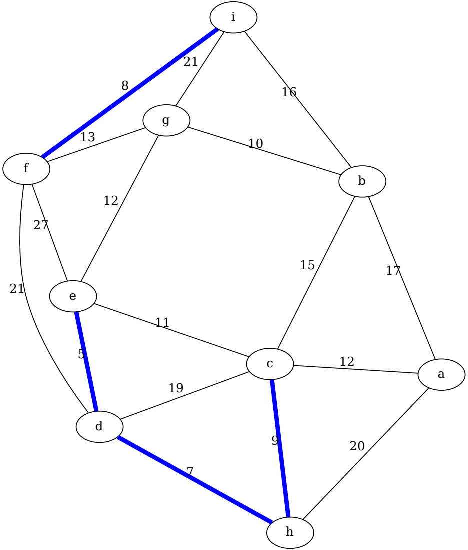
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	←
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



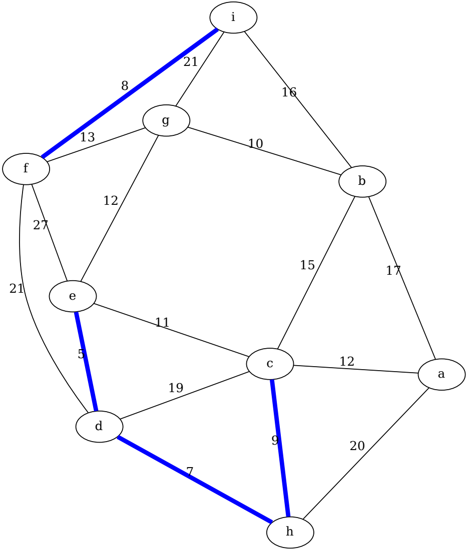
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	←
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



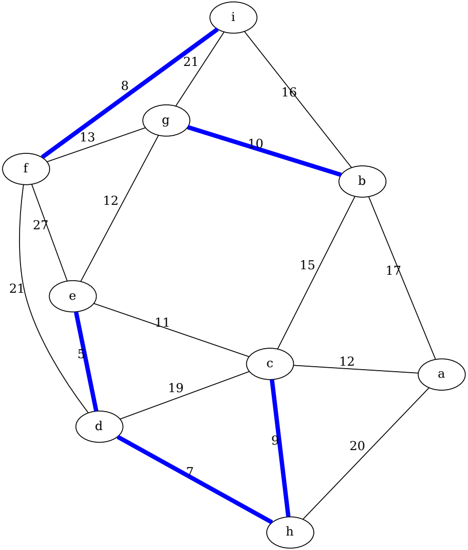
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	←
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



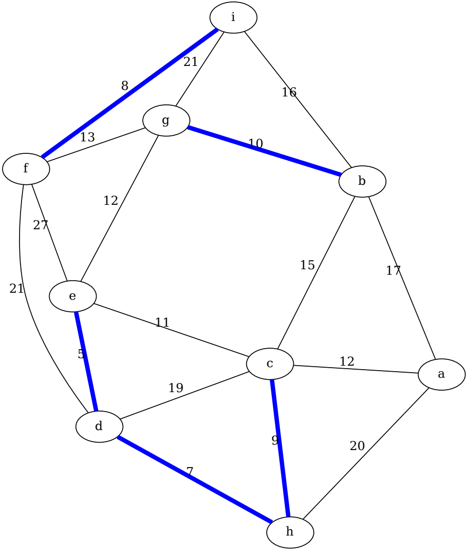
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	←
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



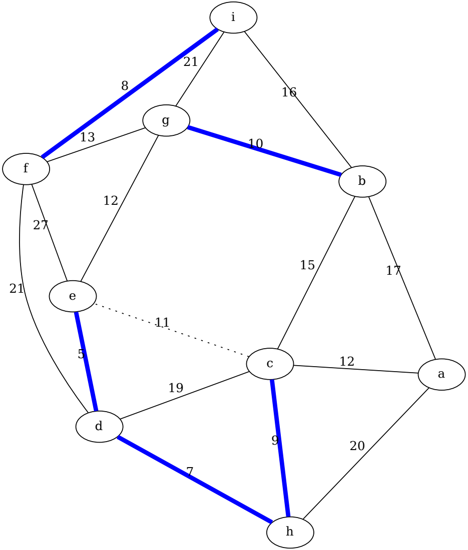
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	←
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



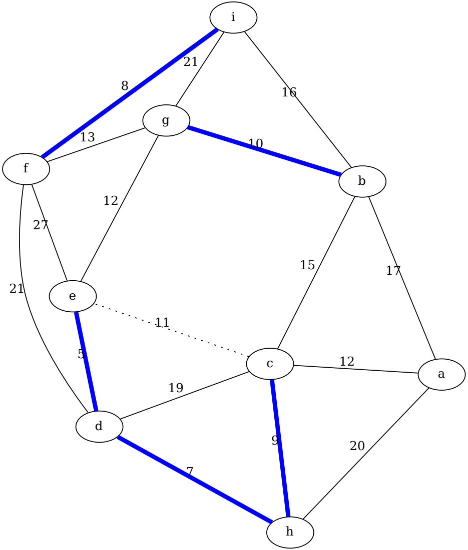
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	←
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



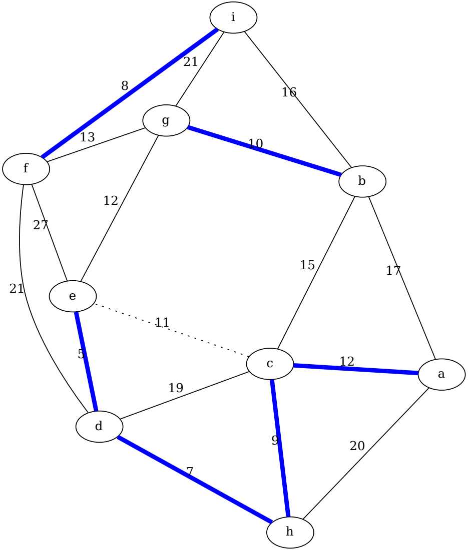
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	←
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



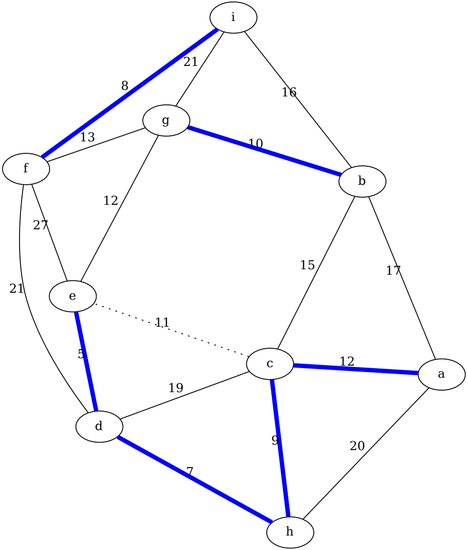
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	←
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



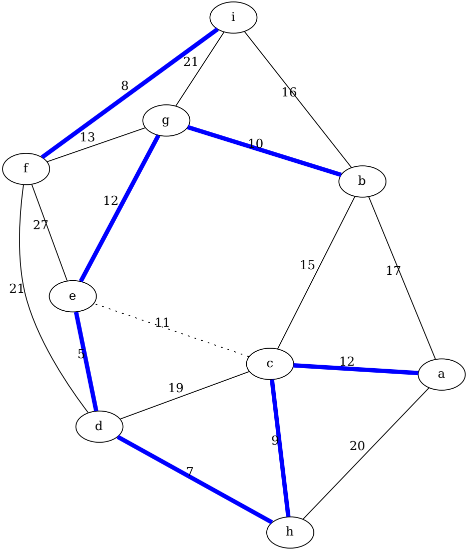
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	←
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



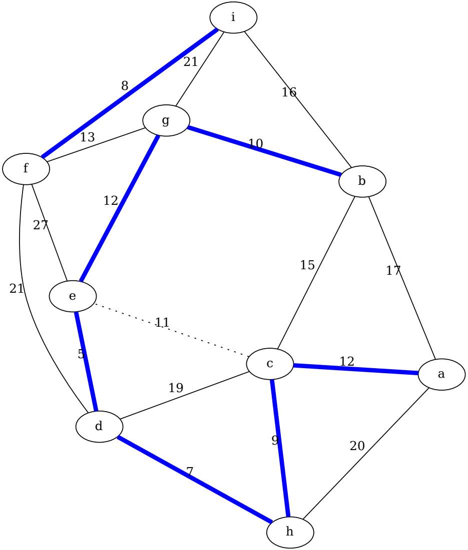
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	←
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



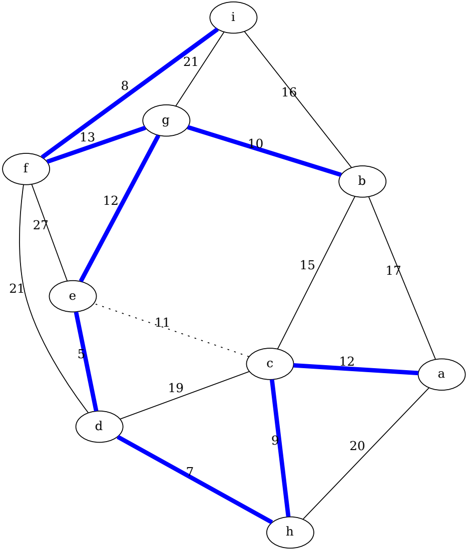
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	←
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



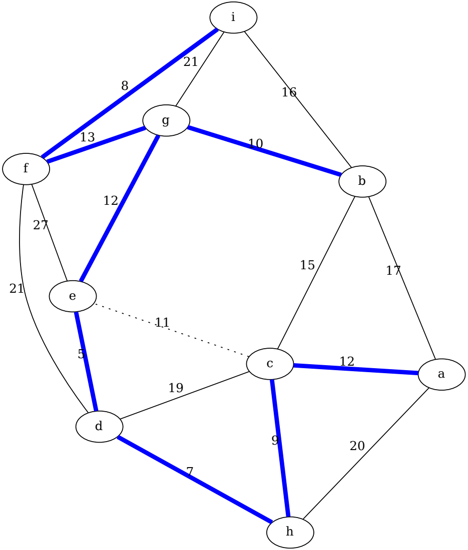
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	←
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



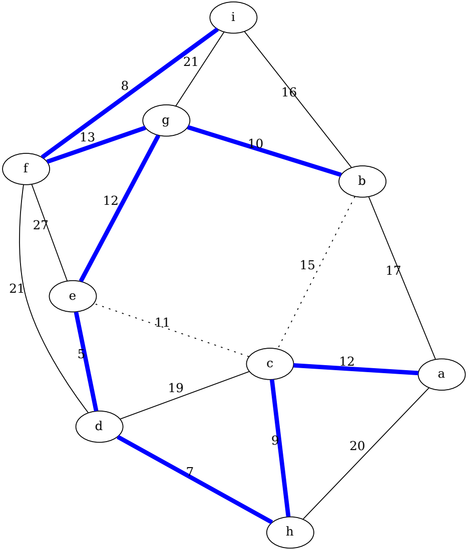
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	$\leftarrow$
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge $\longrightarrow$ not yet considered.
- Blue edge $\longrightarrow$ tree edge.
- Dotted edge $\longrightarrow$ non-optimal edge.

Example



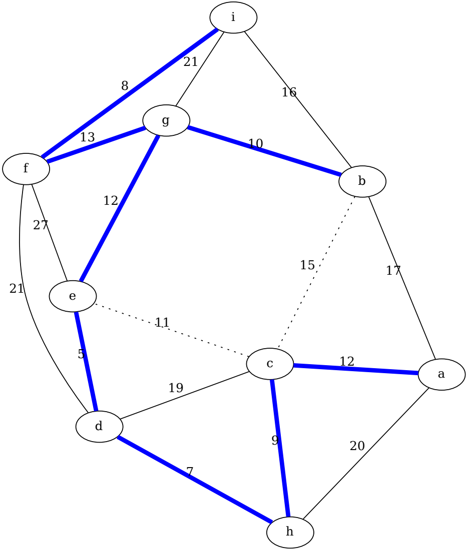
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	←
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



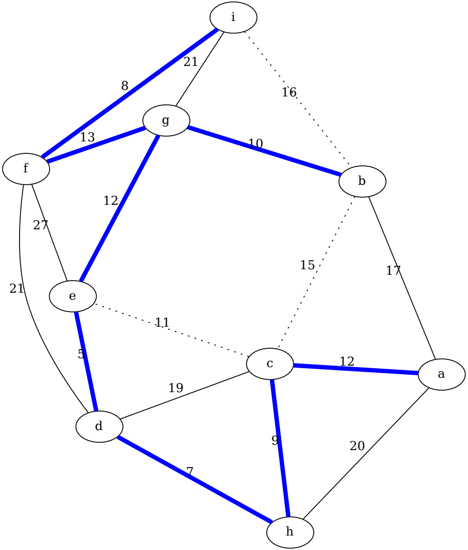
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	←
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



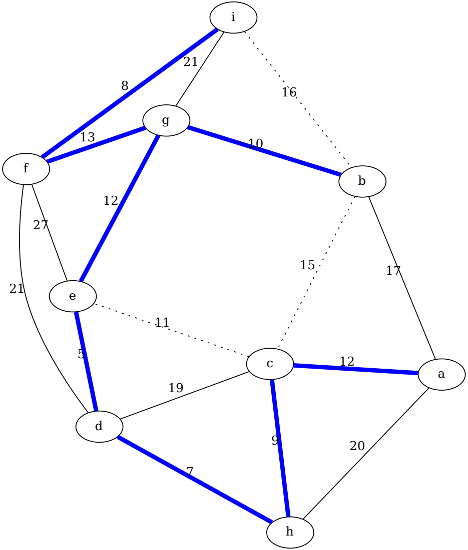
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	←
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



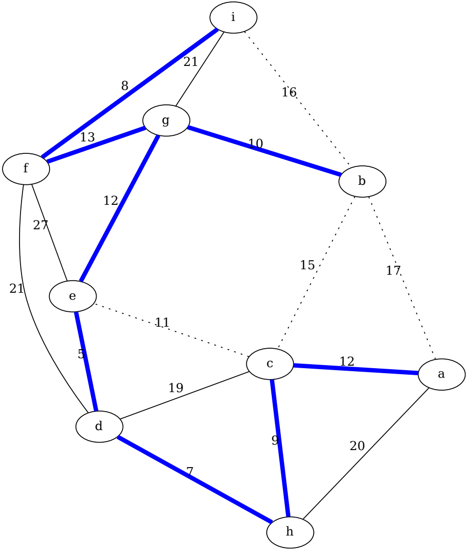
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	←
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



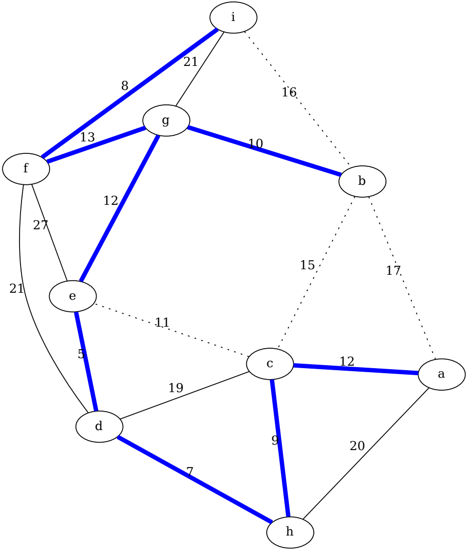
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	←
cd	19	
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



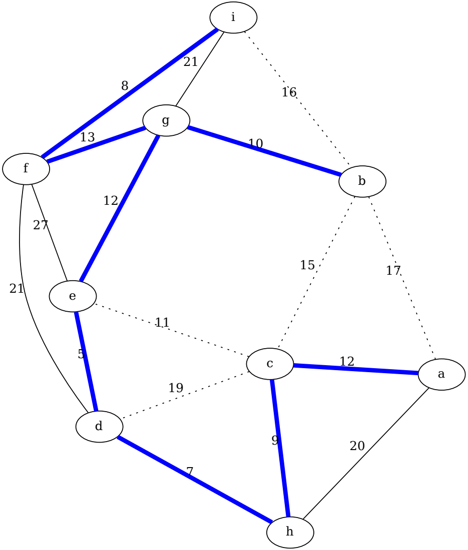
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	←
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



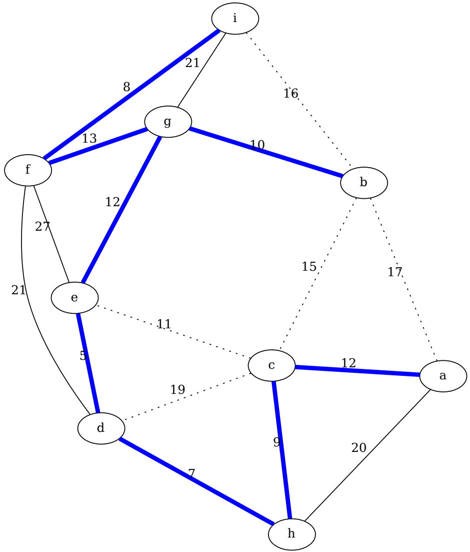
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	←
ah	20	
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



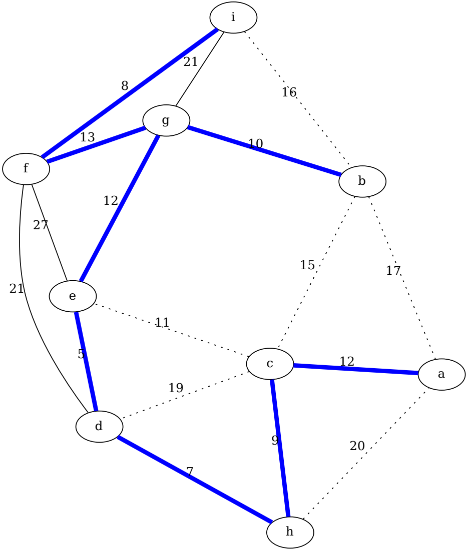
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	←
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



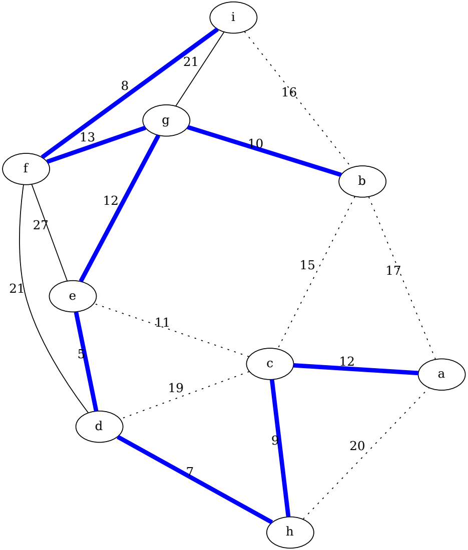
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	←
df	21	
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



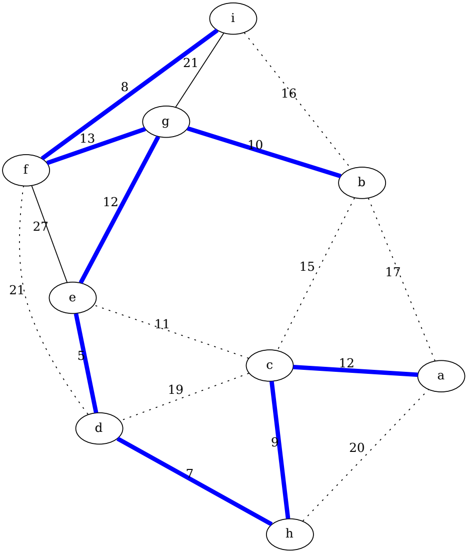
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	←
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



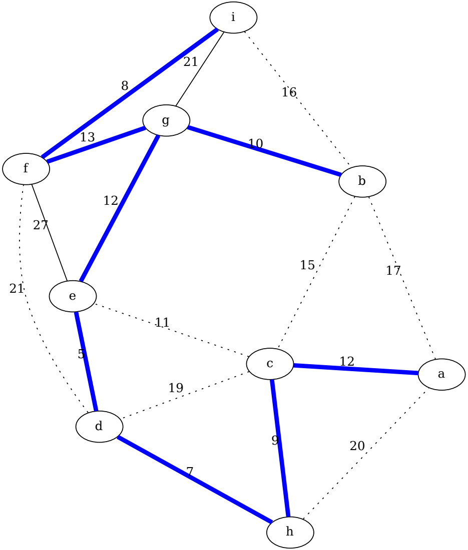
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	←
gi	21	
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



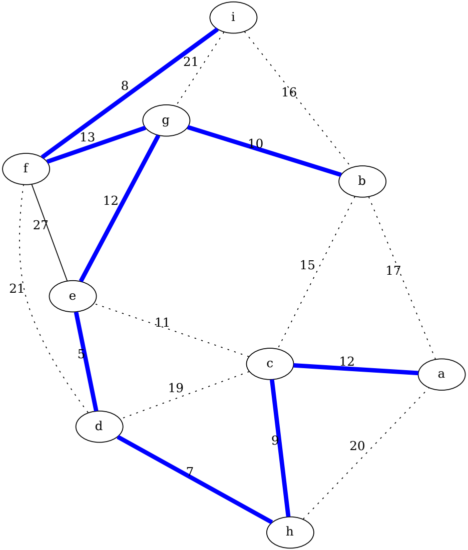
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	←
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



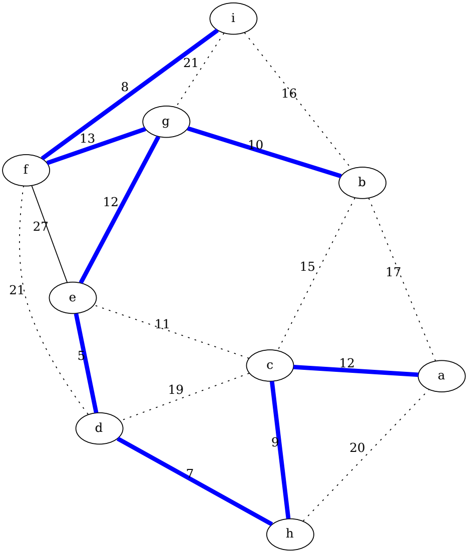
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	←
ef	27	

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



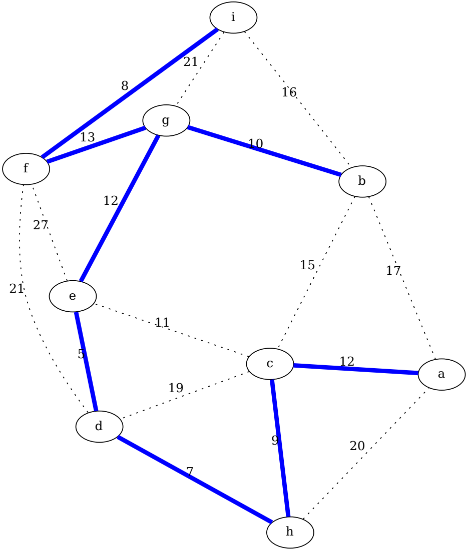
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	←

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Example



Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
cd	19	
ah	20	
df	21	
gi	21	
ef	27	←

Legend:

- Black edge → not yet considered.
- Blue edge → tree edge.
- Dotted edge → non-optimal edge.

Proof of correctness

Theorem

Kruskal's algorithm outputs a spanning tree of minimum weight.

Proof of correctness

Theorem

Kruskal's algorithm outputs a spanning tree of minimum weight.

Proof.

- Easy to see that algorithm outputs some spanning tree
  - We consider all edges, only refuse one if we already have a path between its endpoints.
- Let $E_i = \{e_1, \ldots, e_i\}$ and $S_i \subseteq E_i$ be the set of edges selected by Kruskal from E_i.
- Claim: for all i, there exists a minimum spanning tree T^* with $S_i \subseteq T^*$.
- If claim is correct $\Rightarrow$ Kruskal is correct.

□

Lemma

For all i, there exists a minimum spanning tree T^* with $S_i \subseteq T^*$.

Lemma

For all i, there exists a minimum spanning tree T^* with $S_i \subseteq T^*$.

Proof.

Proof by induction:

- $i = 0$, easy.
- $i = 1$, easy(?)

Lemma

For all i, there exists a minimum spanning tree T^* with $S_i \subseteq T^*$.

Proof.

Proof by induction:

- $i = 0$, easy.
- $i = 1$, easy(?)
  - If $e_1 = xy$, set $S = \{x\}$ and e_1 is light for S.

Lemma

For all i, there exists a minimum spanning tree T^* with $S_i \subseteq T^*$.

Proof.

Proof by induction:

- $i = 0$, easy.
- $i = 1$, easy(?)
  - If $e_1 = xy$, set $S = \{x\}$ and e_1 is light for S.
- Suppose true for i, show for $i + 1$:
- If Kruskal does not select e_{i+1}, done!
- If it does and $e_{i+1} = xy$, let C, C' be the components of x, y in $G[S_i]$.
  - C, C' are distinct, because Kruskal accepted e_{i+1}.
- Observation: e_{i+1} is light for C, all edges of S_i have 0 or 2 endpoints in C.

□

Complexity Analysis

- Step 1: sort edges $\Rightarrow O(m \log n)$
- Repeat m times, check if adding $e = xy$ adds a cycle
  - Need to check if path $x \to y$ already exists.
  - Can be done in $O(m)$ for connected graphs.
- Complexity: $O(m^2)$, much worse than Prim!
- Can we do better?

Union-Find

We need two operations:

- (**Find**): Given two vertices x, y, check if x, y are already in same connected component.
- (**Union**): Merge the components of two given vertices x, y.

Where we need these:

- Find: test if adding the edge $e = xy$ creates a cycle.
- Union: if not, we add e to the graph, so the two components become one.

Union-Find naïvely

- Create an array $Comp[1 \dots n]$ such that $C[i]$ is the component number of vertex i.
- Initially $Comp[i] = i$ for all $i \in \{1, \dots, n\}$.
- Find: check if $Comp[i] == Comp[j]$
- Union:

```
procedure UNION(i,j)
    c2 ← Comp[j]
    for k = 1 to n do
        if Comp[k] == c2 then
            Comp[k] ← Comp[i]
        end if
    end for
end procedure
```

Analysis of naïve solution

- Find takes constant time (good!)
- Union takes $O(n)$ time (bad!)
- Total complexity: $O(mn)$
  - Slightly better than running a connectivity algorithm each time ($O(m^2)$)
- Can we do better?

Faster Union-Find

- We define two attributes for each vertex v
  - The leader of v, denoted $v.\ell$ is (supposed to be) a vertex (possibly v itself) that is the most important in v's component.
  - The rank of v, denoted $v.r$ gives an idea of the size of v's component.
- Initially, $v.\ell \leftarrow v$, $v.r \leftarrow 1$ for all v.

Faster Union-Find

- We define two attributes for each vertex v
  - The leader of v, denoted $v.\ell$ is (supposed to be) a vertex (possibly v itself) that is the most important in v's component.
  - The rank of v, denoted $v.r$ gives an idea of the size of v's component.
- Initially, $v.\ell \leftarrow v$, $v.r \leftarrow 1$ for all v.

Key ideas:

- To check if x, y are in the same component, we check if they have the same leader.
- When we merge, the leader of the new component is the leader of the larger component (based on rank).

Algorithms – Find

procedure FIND-SET(x) ▷ Returns the leader of x's component
  if $x.\ell == x$ **then** ▷ x is a leader
    Return x
  else
    Return Find-Set($x.\ell$)
  end if
end procedure

Algorithms – Find

procedure $\text{Find-Set}(x)$ ▷ Returns the leader of x's component
  if $x.\ell == x$ **then** ▷ x is a leader
    Return x
  else
    Return Find-Set($x.\ell$)
  end if
end procedure

To check if $e = xy$ creates a cycle check if Find-Set(x)==Find-Set(y).

Algorithms – Union

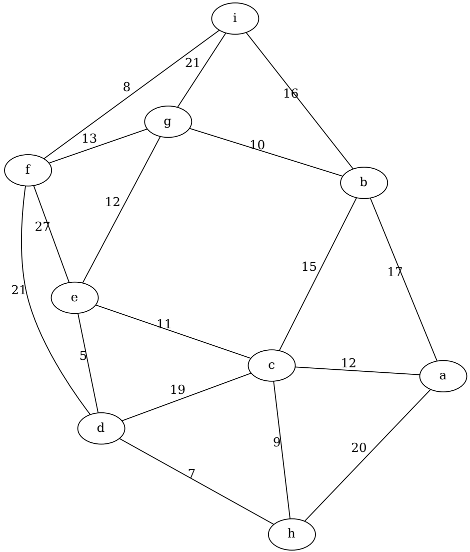
```
procedure UNION(x, y)                 ▷ Assume x, y are leaders
    if x.r > y.r then
        y.ℓ ← x                       ▷ y not a leader anymore
    else
        x.ℓ ← y
        if x.r == y.r then
            y.r++                     ▷ Merged two equal rank components
        end if
    end if
end procedure
```

Algorithms – Union

```
procedure UNION(x, y)                          ▷ Assume x, y are leaders
    if x.r > y.r then
        y.ℓ ← x                                ▷ y not a leader anymore
    else
        x.ℓ ← y
        if x.r == y.r then
            y.r++                              ▷ Merged two equal rank components
        end if
    end if
end procedure
```

To merge two arbitrary z, w, call Union(Find-Set(z),Find-Set(w)).

Example



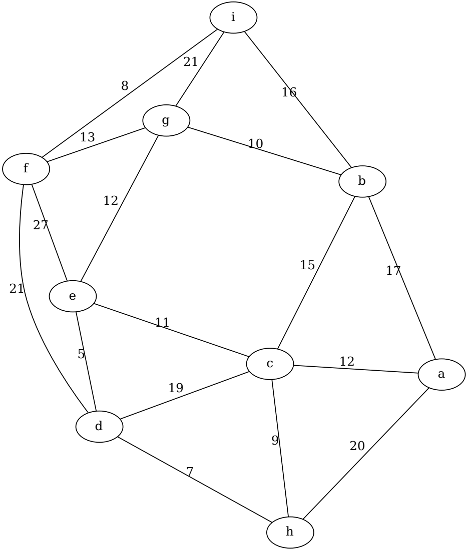
Sorted list of edges:

Edge	Weight
de	5
dh	7
fi	8
ch	9
bg	10
ce	11
ac	12
eg	12
fg	13
bc	15
bi	16
ab	17
...	

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	1
e	e	1
f	f	1
g	g	1
h	h	1
i	i	1

Example



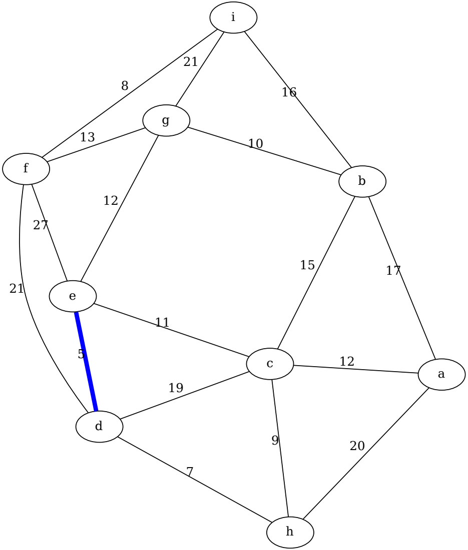
Sorted list of edges:

Edge	Weight	
de	5	←
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	1
e	e	1
f	f	1
g	g	1
h	h	1
i	i	1

Example



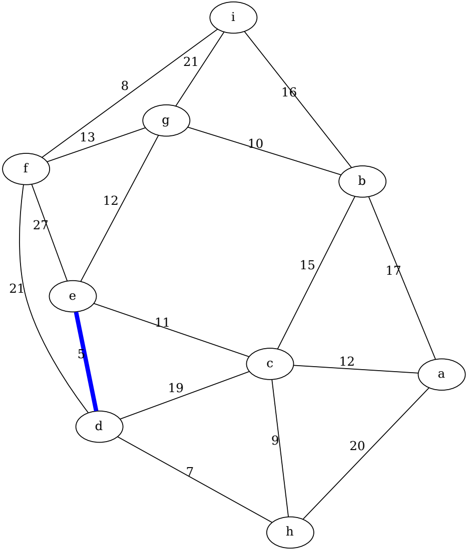
Sorted list of edges:

Edge	Weight	
de	5	←
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	2
e	d	1
f	f	1
g	g	1
h	h	1
i	i	1

Example



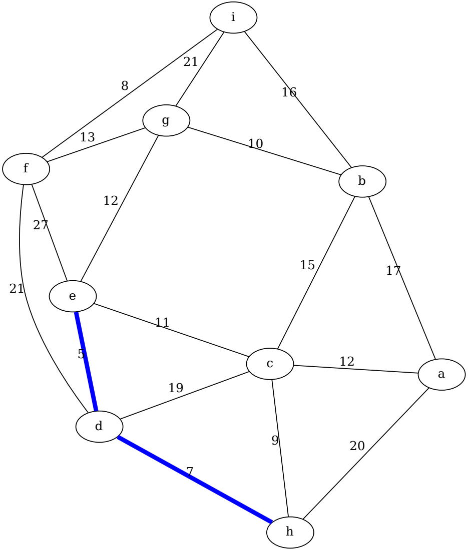
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	←
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	2
e	d	1
f	f	1
g	g	1
h	h	1
i	i	1

Example



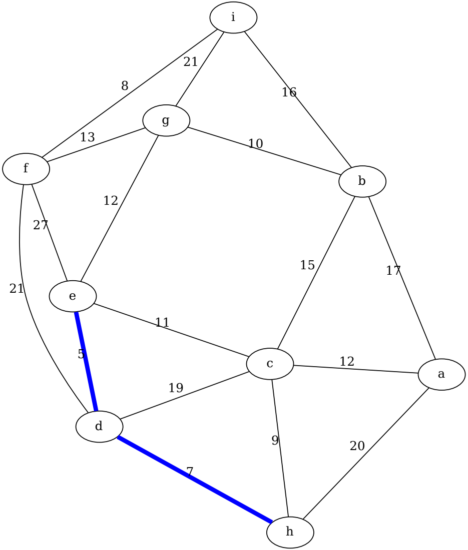
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	←
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	2
e	d	1
f	f	1
g	g	1
h	d	1
i	i	1

Example



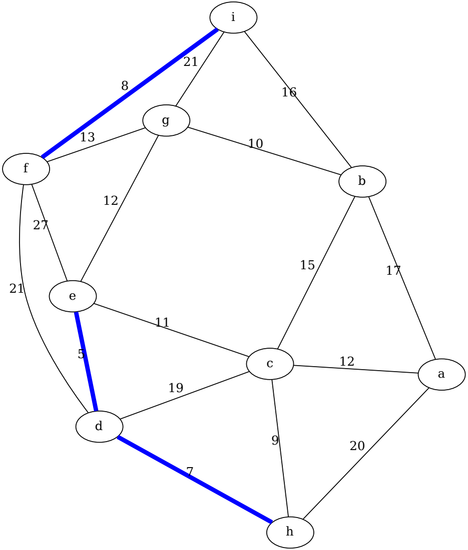
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	←
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	2
e	d	1
f	f	1
g	g	1
h	d	1
i	i	1

Example



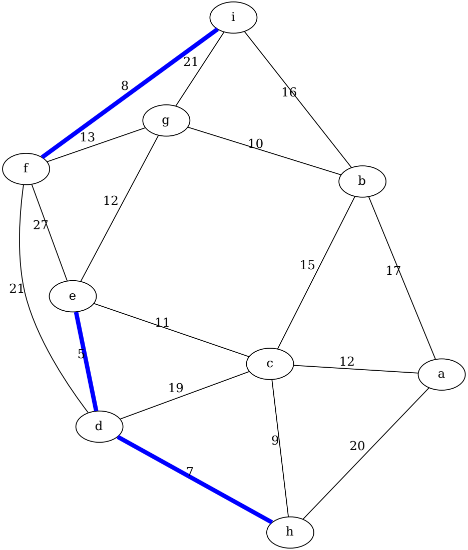
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	←
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	2
e	d	1
f	f	2
g	g	1
h	d	1
i	f	1

Example



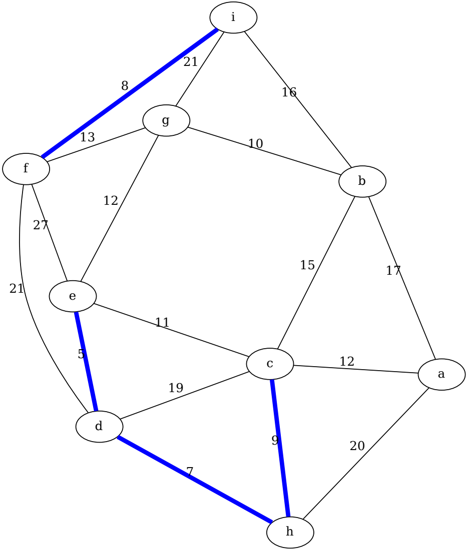
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	←
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	c	1
d	d	2
e	d	1
f	f	2
g	g	1
h	d	1
i	f	1

Example



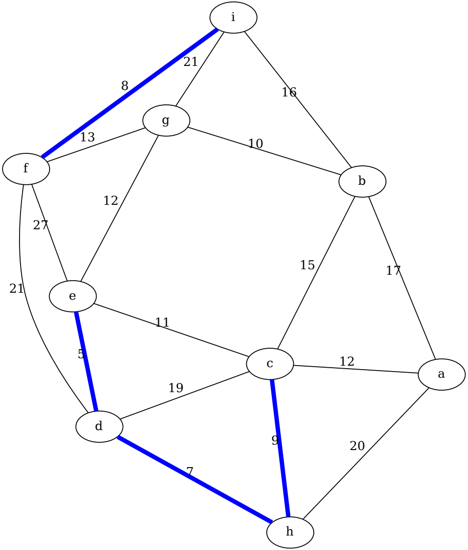
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	←
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	d	1
d	d	2
e	d	1
f	f	2
g	g	1
h	d	1
i	f	1

Example



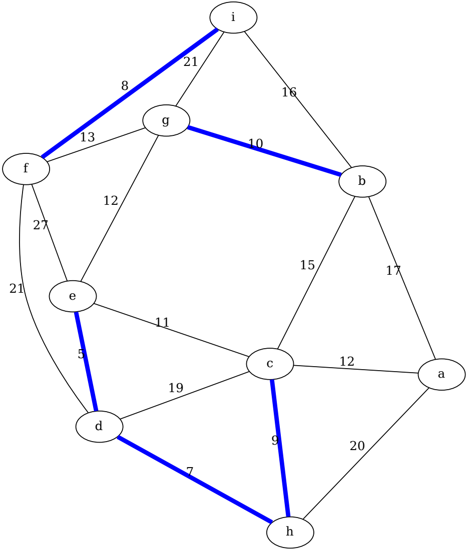
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	←
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	1
c	d	1
d	d	2
e	d	1
f	f	2
g	g	1
h	d	1
i	f	1

Example



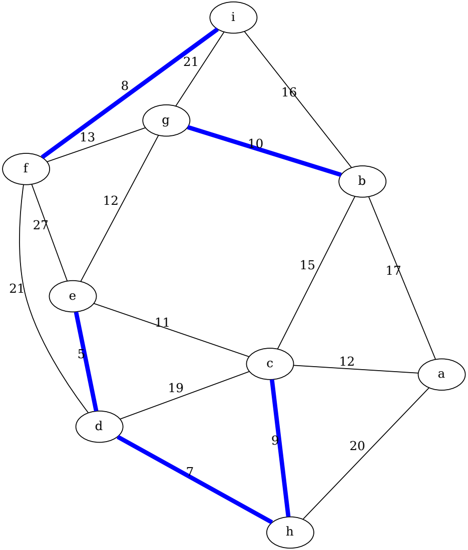
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	←
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	2
c	d	1
d	d	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



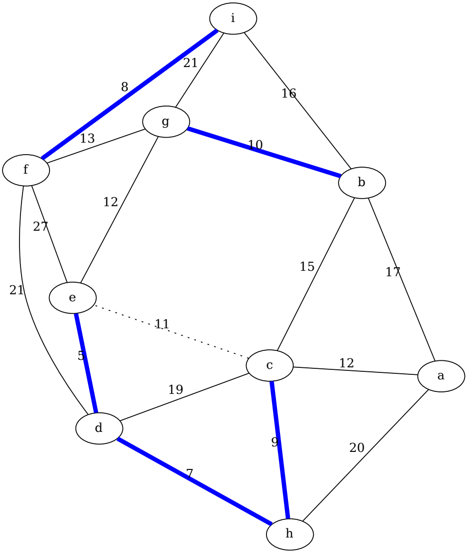
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	←
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	2
c	d	1
d	d	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



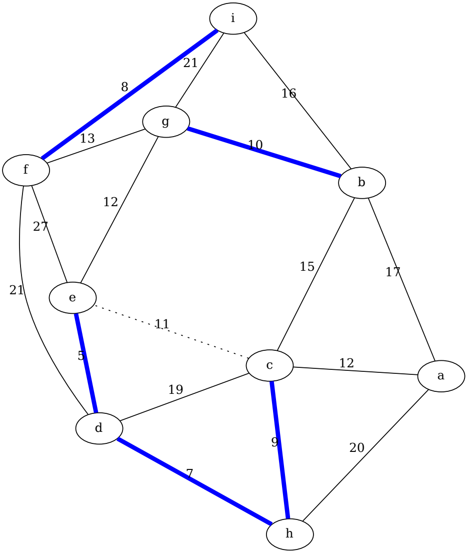
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	←
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	2
c	d	1
d	d	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



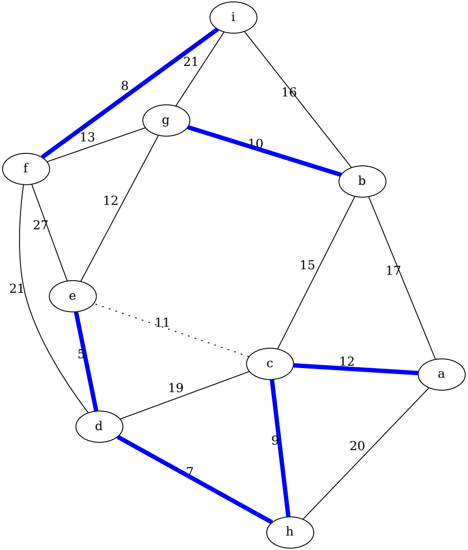
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	←
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	a	1
b	b	2
c	d	1
d	d	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



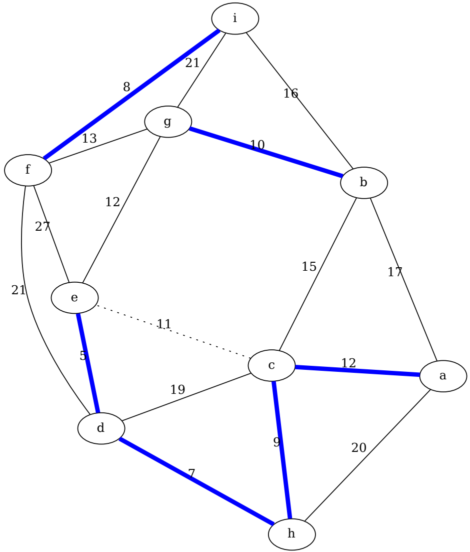
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	←
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	2
c	d	1
d	d	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



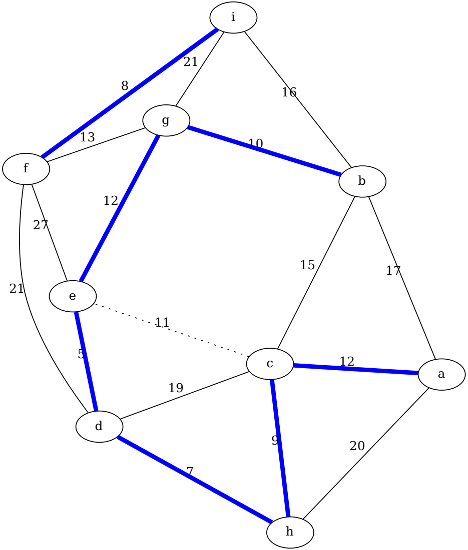
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	←
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	2
c	d	1
d	d	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



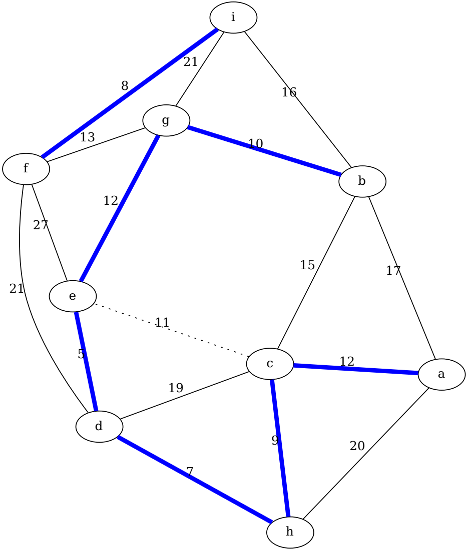
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	←
fg	13	
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



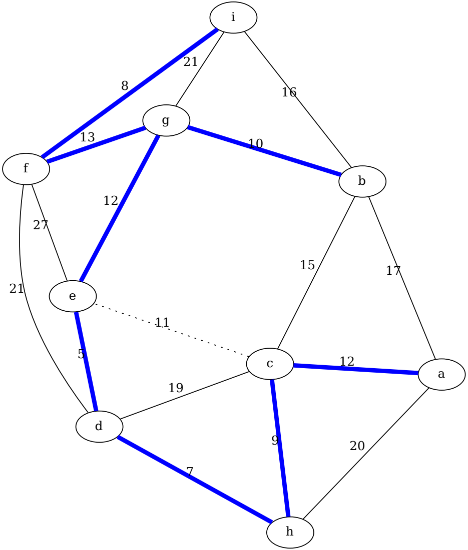
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	←
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	f	2
g	b	1
h	d	1
i	f	1

Example



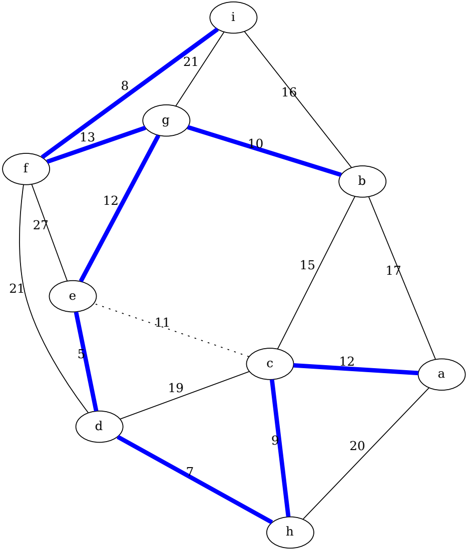
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	←
bc	15	
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Example



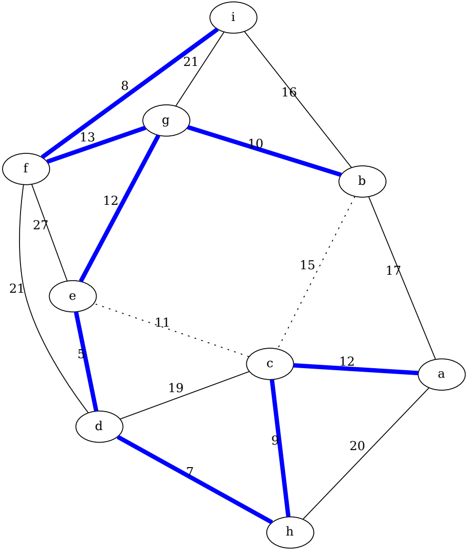
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	←
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Example



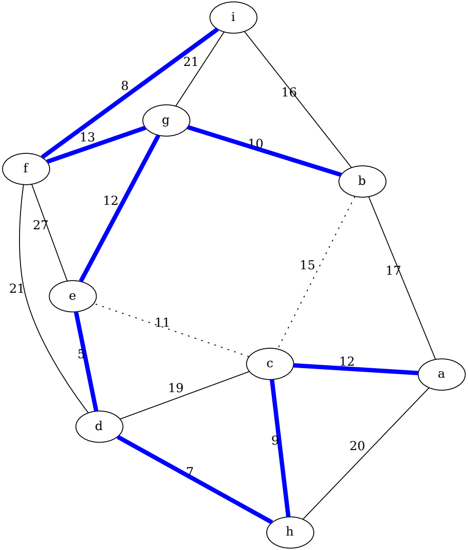
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	←
bi	16	
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Example



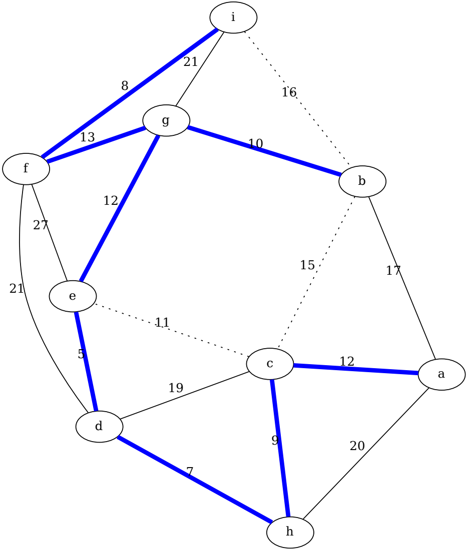
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	←
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Example



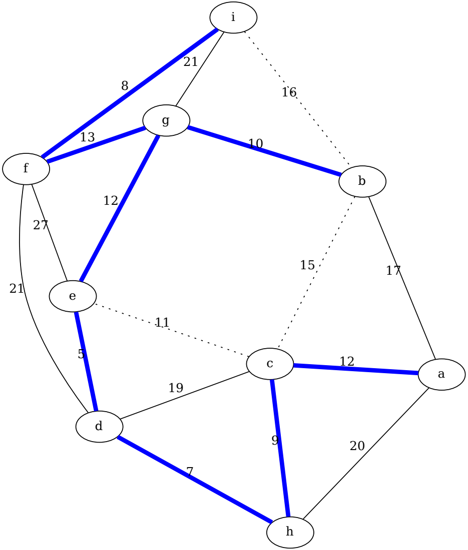
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	←
ab	17	
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Example



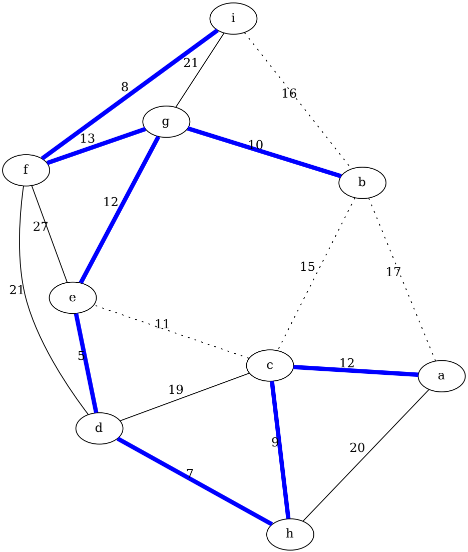
Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	←
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Example



Sorted list of edges:

Edge	Weight	
de	5	
dh	7	
fi	8	
ch	9	
bg	10	
ce	11	
ac	12	
eg	12	
fg	13	
bc	15	
bi	16	
ab	17	←
...		

Ranks and leaders:

Vertex	Leader	Rank
a	d	1
b	b	3
c	d	1
d	b	2
e	d	1
f	b	2
g	b	1
h	d	1
i	f	1

Analysis

Lemma

All vertices have rank at most $O(\log n)$.

Lemma

Find-Set takes time $O(\log n)$.

Theorem

Kruskal's algorithm runs in time $O(m \log n)$.

Lemma

All vertices have rank at most $O(\log n)$.

Lemma

All vertices have rank at most $O(\log n)$.

Proof.

Equivalently: if a vertex has rank i, its component has at least 2^{i-1} vertices.

- For $i = 1$, correct.
- A vertex attains rank $i + 1$ if it had rank i and it is the leader of a component merged with another one whose leader has rank i.
- Inductive hypothesis: both components have $\geq 2^{i-1}$ vertices, so $\geq 2^i$ vertices in total.

$\square$

Lemma

Find-Set takes time $O(\log n)$.

Lemma

Find-Set takes time $O(\log n)$.

Proof.

- Observation: if $x.\ell \neq x$, then $x.\ell.r > x.r$ (my leader has higher rank)
  - Proof by induction, whenever we modify a leader the condition is true.
- Find-Set either returns immediately, or calls itself on a vertex with higher rank.
- Since ranks are at most $O(\log n)$, the lemma follows.

$\square$

Theorem

Kruskal's algorithm runs in time $O(m \log n)$.

Theorem

Kruskal's algorithm runs in time $O(m \log n)$.

Proof.

- Repeat m times:
- Run two Find-Set calls for edge $e = xy$ to find the leaders of the components of x, y
- If leaders are the same, continue to next edge
- If not, treat the two leaders in $O(1)$ time.
- Total complexity: $O(m \log n)$

□

Going further

A further improvement is possible:

procedure $\textsc{Find-Set}(x)$ ▷ Returns the leader of x's component
  if $x.\ell == x$ **then** ▷ x is a leader
    Return x
  else
    Return Find-Set($x.\ell$)
  end if
end procedure

Going further

A further improvement is possible:

procedure FIND-SET(x) ▷ Returns the leader of x's component
  if $x.\ell == x$ **then** ▷ x is a leader
    Return x
  else
    $x.\ell \leftarrow$ Find-Set($x.\ell$)
    Return $x.\ell$ ▷ Path Compression
  end if
end procedure

- The improved version actually runs in $O(m\alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function (significantly less than $\log n$).
- Analysis too complicated for this course.
- So, if edges are pre-sorted, Kruskal is **slightly faster** than Prim!

Summary

Minimum Spanning Tree Algorithms:

- Input: Connected undirected graph G with edge weights
- Prim's: greedily extend the current tree
  - Complexity: $O(m \log n)$ (using min-heaps)
- Kruskal's: greedily extend the current forest
  - Complexity: $O(m \log n)$ (using Union-Find with ranks)
  - Complexity: $O(m\alpha(n))$ (using Union-Find with ranks and path compression, if edges are pre-sorted)